Core Java: Reflection API

Persistent University

Key learning points:

- Understand reflection API.
- Understand how to use Java reflection for
  - Classes
  - Modifiers
  - Packages
  - Interfaces
  - Constructors
  - Fields
  - Methods

Persistent

What is Reflection?

- Java Reflection is a process of examining or modifying the run time behavior of a class. This is achieved at runtime.
- The **java.lang.Class** class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

- The java.lang and **java.lang.reflect** packages provide classes for java reflection.

Where is it used?

- The Reflection API’s usage is mainly in
  - IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans etc.
  - Debugger
  - Test Tools
  - Frameworks as Junit, Spring, Struts, Hibernate etc.

java.lang.Class class

- The **java.lang.Class** class performs mainly two tasks:
  - provides methods to get the metadata of a class at run time.
  - provides methods to examine and change the run time behavior of a class.

Java reflection - Classes

- The ways to fetch information about a class are
  - Use of **forName()** method of Class class
  - Use of **getClass()** method of Object class
  - Use of **.class** syntax

The forName() method

- This method is used to load the class dynamically.
- It returns the instance of Class class.
- It should be used if fully qualified name of class is known.
- This method cannot be used for primitive types.

```
class Demo{}

class ReflectionTest{
    public static void main(String args[]) throws Exception{
            Class aClass=Class.forName(“Demo");
            System.out.println(aClass.getName());
    }
}
```

```
//Output :
            Demo
```

The getClass() method of Object class

- This method returns the instance of Class class. It should be used if the type of the class is known.
- Can be used with primitives.

```
class Demo{}

class Test{
  void printName(Object obj){
                Class aClass=obj.getClass();
                System.out.println(aClass.getName());
  }
  public static void main(String args[]){
                 Demo demo = new Demo();
                 Test t=new Test();
                 t.printName(demo);
 }
}

// Output
Demo
```

The .class syntax

- If a type is available but there is no instance, then it is possible to obtain a Class by appending ".class" to the name of the type.
- It can be used for primitive data type also.

```
class Test{
  public static void main(String args[]){
   Class aClass = boolean.class;
   System.out.println(aClass.getName());

   Class c2 = Test.class;
   System.out.println(c2.getName());
 }
}

// Output
  boolean
  Test
```

Java reflection – Modifiers

- We can access the modifiers of a class via the Class object.
- The modifiers are represented in an int. These modifiers can be checked using various methods of class **java.lang.reflect.Modifier**.

```
Modifier.isAbstract(int modifiers)
Modifier.isFinal(int modifiers)
Modifier.isInterface(int modifiers)
Modifier.isNative(int modifiers)
Modifier.isPrivate(int modifiers)
Modifier.isProtected(int modifiers)
Modifier.isPublic(int modifiers)
Modifier.isStatic(int modifiers)
Modifier.isStrict(int modifiers)
Modifier.isSynchronized(int modifiers)
 Modifier.isTransient(int modifiers)
 Modifier.isVolatile(int modifiers)
```

```
class Demo{}

public class ReflectionTest {
    public static void main(String[] args) {
            try {
            Class aClass = Class.forName("Demo");
            int modifier = aClass.getModifiers();
            System.out.println(modifier);
            } catch (ClassNotFoundException e) {
                    e.printStackTrace();
            }
    }
}
```

Java reflection – Package

- The information about the package from a Class object can also be obtained.
- **java.lang.Package** class object fetches this information.

Output :

package com.persistent

```
package com.persistent;
public class Demo{}
                ----------------------------------------------------------
public class ReflectionTest {
  public static void main(String[] args) {
    try {
       Class aClass = Class.forName("com.persistent.Demo");
       Package p = aClass.getPackage();
       System.out.println(p);
    } catch (ClassNotFoundException e) {
                e.printStackTrace();
    }}}
// Output
Package com.persistent
```

Java reflection – Methods

- A class can contain various methods. The **getMethods()** method returns the method names of the class.
- The method names from current class and super class method names (which are accessible to this class) will be stored in an array of type **java.lang.reflect.Method** class.

```
package com.persistent;
public class Demo {
    public void method1(){}
    public void method2(int a){}
     void method3(int a, int b){}
}
```

Persistent

Java reflection – Methods….

- A class can contain various methods. The **getMethods()** method returns the method names of the class.
- The method names from current class and super class method names (which are accessible to this class) will be stored in an array of type **java.lang.reflect.Method** class.

```
public class Demo {
   public void method1(){}
   public void method2(int a){}
    void method3(int a, int b){}
}

import java.lang.reflect.Method;
public class ReflectionTest {
public static void main(String[] args) {
try {
   Class aClass = Class.forName("Demo");
   Method methods[] = aClass.getMethods();
   for (Method methodName : methods) {
                System.out.println(methodName);
  }
} catch (ClassNotFoundException e) {
                e.printStackTrace();
} }  }
```

Java reflection – Fields

- A class can contain various fields. The **getFields()** method returns the field names of the class.
- The *field names of accessible variables* will be stored in an array of type **java.lang.reflect.Field** class.

```
import java.lang.reflect.Field;
public class ReflectionTest {
public static void main(String[] args) {
try {
            Class aClass = Class.forName(“com.persistent.Demo");
            Field field[] = aClass.getFields();
            for (Field fields : field) {
               System.out.println(fields);
            }
} catch (ClassNotFoundException e) {
            e.printStackTrace();
} }  }
```

Summary :

- With this we have come to an end of our session, where we discussed about ….
  - Reflection API and how we can examine the classes at runtime.
- At the end of this session, we see that you are now able to answer following questions:
  - What is Reflection API?
  - How to make use of java.lang.Class class and reflection API provided by java to fetch the details of a class at runtime?

Appendix

References

Thank you

Reference Material : Websites & Blogs

- http://tutorials.jenkov.com/java-reflection/index.html
- http://www.javatpoint.com/java-reflection
- http://www.journaldev.com/1789/java-reflection-example-tutorial

Reference Material : Books

- ***Java Complete Reference***
    - *By Herbert Schildt*

Persistent

Thank you!

Persistent University